# 设计模式笔记总结

----刘世雄 2017/01/20

## 1.解决问题的两种方法：

**分解问题**

**抽象问题**

## 2.设计模式八大原则

### 一 依赖倒置原则：

依赖倒置原则(DIP)：

高层模块(稳定)不应该依赖于底层模块(变化)，二者都依赖于抽象(稳定)

抽象(稳定)不应该依赖于实现的具体细节(变化)，实现细节应该依赖于抽象(稳定)

### 二 开放封闭原则：

开放封闭原则(OCP)：

对扩展开放，对更改封闭。

类模块应该是可扩展的，但是不可修改。

### 三 单一职责原则：

单一职责原则(SRP)：

一个类仅有一个引起它变化的原因。

变化的方向隐含着类的责任。

### 四 Liskov替换原则：

Liskov替换原则(LSP)

子类必须能够替换它们的基类(IS-A)。

继承表达类型抽象。

### 五 接口隔离原则：

不应该强迫客户程序依赖它们不用的方法。

接口应该小而完备。

### 六 优先使用对象组合：

优先使用对象组合，而不是继承

类继承通常为“白箱复用”，对象组合通常为“黑箱复用”。

继承在某种程度上破坏了分装性，子类父类耦合度较高。

而对象组合则只要求被组合的对象具有良好定义接口，耦合度低。

### 七 分装变化点：

分装变化点

使用封装来创建对象之间的分界层，让设计者可以在分界一侧进行修改，而不会对另一侧产生不良的影响，从而实现层次间的松耦合。

### 八 针对接口编程：

针对接口编程，而不是针对实现编程

不将变量类型声明为某个特定的具体类，而是声明为某个接口。

客户程序无需获知对象的具体类型，只需要知道对象所具有的接口。

减少系统中各部分的依赖关系，从而实现“高内聚，松耦合”的类型设计方案。

**设计模式最大的特点就是在稳定和变化点中需求隔离点，从而分离变化点管理变化点，稳定中有变化**

## 3.面向对象中复用的概念：

指的是二进制单位的复用（就是源代码编译，测试之后不再改动，编译单位的复用性），不是指源代码片段中的复用。

## 4.注意点：

* 1.使用设计模式的时候，需要分析稳定点和变化点，在变化点处使用设计模式。
* 2.软件的分析不要静态的分析，而是采用动态的分析，从时间轴上去分析，软件的需求是不断的发生变化。
* 3.多继承使用的正确姿势，一个主类，多个接口。
* 4.组合优于继承，继承可能会造成代码的重复

## 5.重构关键技法：

* **静态** 🡪 **动态**
* **早绑定** 🡪 **晚绑定**
* **继承** 🡪 **组合**
* **编译时依赖** 🡪 **运行时依赖**
* **紧耦合** 🡪 **松耦合**

## 6.设计模式

### 1.组件协作模式

现代软件专业分工之后的第一个结果是“框架和应用成绩的划分”，“组件协作”模式通过晚期绑定，来实现框架和应用程序之间的松耦合，是二者之间协作时常用的模式。

#### 模板方法（Template method）

动机（出现的条件）：

在软件构建的过程中，对于某一项任务，它常常有稳定的操作结构，但各个子步骤却有很多改变的需求，或者由于固有的原因（比如框架和应用之间的关系）而无法和任务的整体结构同时实现。

模式定义：

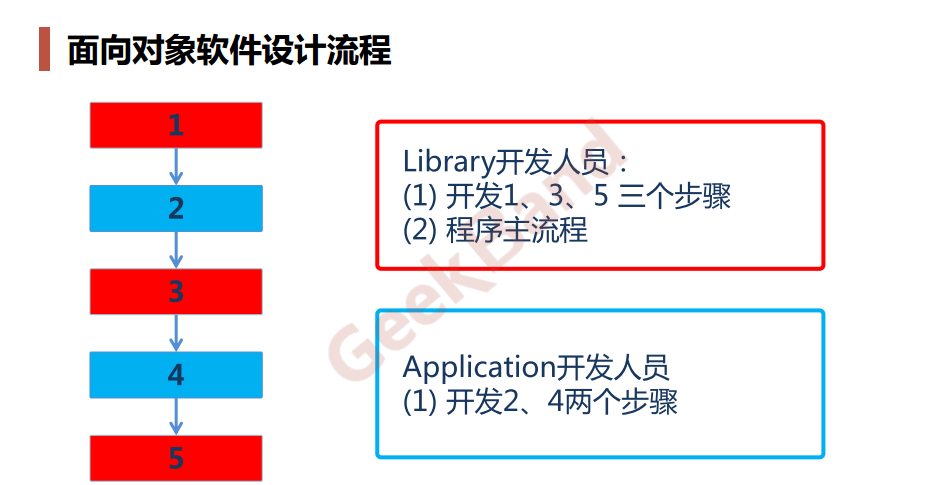
定义一个操作中的算法骨架（稳定），将一些步骤延迟（变化）到子类中实现。Template Method使得子类不改变算法的结构（稳定），就可以重定义（override）该算法的某些特定步骤。

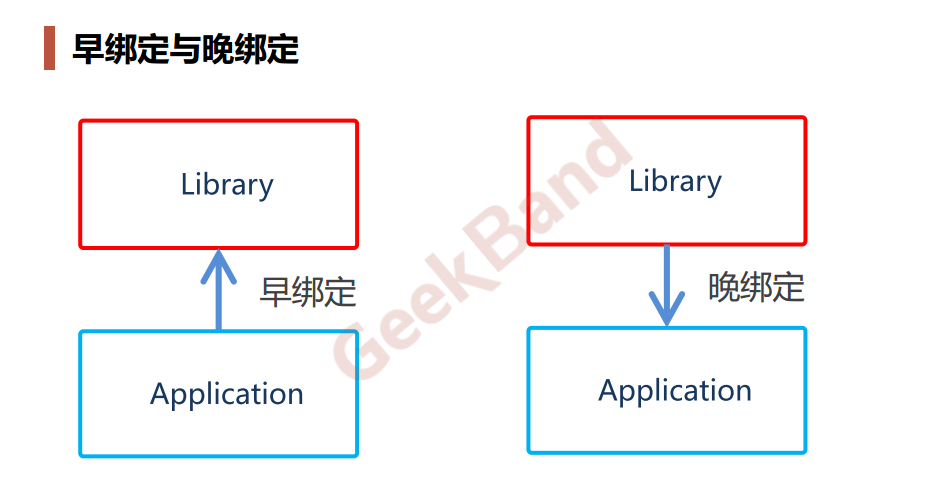
----《设计模式》GoF

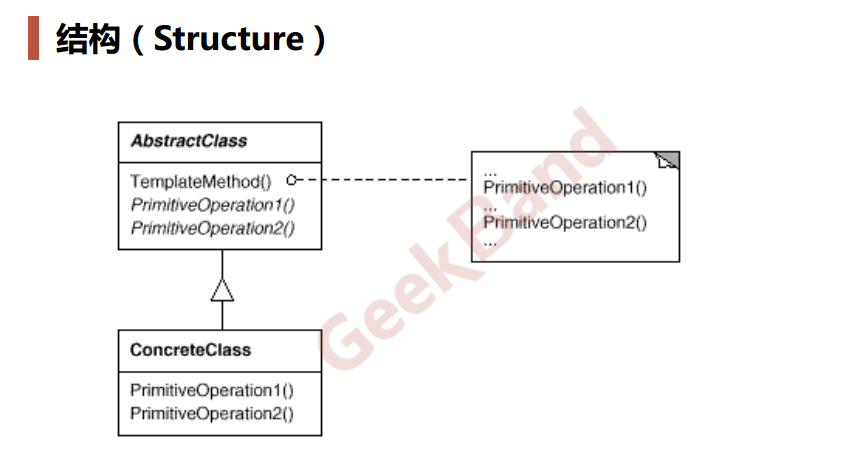
要点总结：

* Template Method 模式是一种非常基础性的设计模式，在面向对象系统中有着大量的引用。它用最简洁的机制（虚函数的多态性）为很多应用程序框架提供了灵活的扩展点，是代码复用方面的最基本实现结构。
* 除了可以灵活应对子步骤的变化外，“不要调用我，让我来调用你”的反向控制结构是Template Method的典型应用。
* 在具体的实现方面，被Template Method调用的虚方法可以具有实现，也可以没有任何的实现（抽象方法，纯虚方法），但一般推荐将它们设置为protected方法。









可以考虑利用模板方法重写关于http的请求代码，以便做成代码库方便以后的开发项目直接拿来用。

#### 2.策略模式（Strategy）

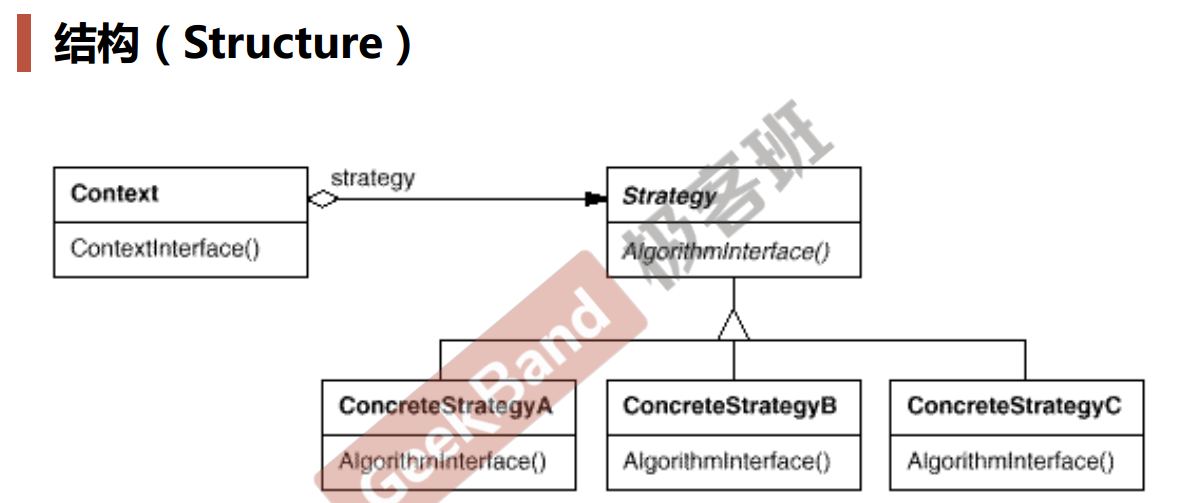
动机：

* 在软件构建过程中，某些对象使用的算法可能多种多样，经常改变，如果将这些算法都编码到对象中，将会使对象变得异常复杂；而且有时候支持不使用的算法也是一个性能负担（将无用的算法代码加载到内存中，会拜拜消耗内存。）
* 将算法与对象本身解耦，从而避免以上问题。

模式定义：

定义一系列算法，把它们一个个封装起来，并且使它们可以相互替换（变化）。该模式使得算法可独立与使用它的客户端程序（稳定）而变化（扩展，子类化）

----《设计模式》 GoF



总结：

* Strategy及其子类为组件提供了一系列可重用的算法，从而可以使子类在运行时候方便地根据需要来在各个算法之间切换
* Strategy模式提供了条件判断以外的另一种选择，消除条件判断语句。含有许多条件判断语句的代码通常都需要Strategy模式。
* 如果Strategy对象没有实例变量，那么各个上下文可以共享同一个Strategy对象，从而节省对象开销。

#### 3.观察者模式（Observe/Events）

体现依赖倒置原则，开放封闭原则

### 2.单一职责模式

#### 1.装饰着模式（Decorator）

#### 2.桥模式（Birdge）